Práctica 4: Proyectar un fichero en memoria

Diseño de Sistemas Operativos – U.L.P.G.C.

David Jesús Horat Flotats
Enrique Fernández Perdomo

Índice

I. Introducción

El objetivo de esta práctica es proyectar un fichero en memoria usando las llamadas al sistema relacionadas, en concreto la mmap y munmap que explicaremos en secciones sucesivas.

Para ello crearemos un fichero de texto con un editor y un programa que realice las siguientes funciones sobre el fichero creado:

1. Abra el fichero
2. Proyecte el fichero en memoria
3. Modifique el contenido del fichero buscando una ristra y reemplazándola
4. Cierre y libere el fichero

Posteriormente se visualizará el contenido del fichero para comprobar que los cambios realizados han sido correctos.

El programa funcionará por línea de comandos, tomando como entrada tres argumentos: el fichero a modificar, la cadena a buscar y la cadena sustituta. Una vez en el programa se mostrarán mensajes que indiquen el estado del proceso y los resultados obtenidos.

II. Explicación de las llamadas de mapeo en memoria

Mmap y munmap

NOMBRE

mmap, munmap - ubica o elimina ficheros o dispositivos en memoria

SINOPSIS

```
#include <unistd.h>
#include <sys/mman.h>

caddr_t mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset);

int munmap(void *start, size_t length);
```

DESCRIPCIÓN

La función mmap intenta ubicar length bytes comenzando en el desplazamiento offset desde el fichero (u otro objeto) espeficicado por fd en memoria, preferiblemente en la dirección start. Esta última dirección es una sugerencia y normalmente se especifica como 0. El lugar donde es ubicado el objeto es devuelto por mmap. El argumento prot describe la protección de memoria deseada. Lo forman los siguientes bits:

PROT_EXEC
Las páginas deben ser ejecutadas.

PROT_READ
Las páginas deben ser leídas.

PROT_WRITE
Las páginas deben ser escritas.

PROT_NONE
Las páginas no pueden ser accedidas.

El parámetro flags especifica el tipo de objeto insertado, las opciones de asociación y si las modificaciones hechas a la copia insertada en memoria son privadas al proceso o son compartidas por otras referencias. Tiene los bits:

MAP_FIXED
No seleccionar una dirección diferente a la especificada. Si la dirección especificada no puede ser utilizada, mmap fallará. Si MAP_FIXED es especificado, start debe ser un múltiplo del tamaño de página. Utilizar esta opción es desaconsejable.

MAP_SHARED
Comparte este área con todos los otros objetos que señalan a este objeto.

MAP_PRIVATE
Crear un área privada "copy-on-write".

Debe especificarse exactamente uno de los parámetros MAP_SHARED o MAP_PRIVATE.

Los tres parámetros anteriores están descritos en POSIX.1b (formalmente POSIX.4). Linux también reconoce MAP_DENYWRITE, MAP_EXECUTABLE y MAP_ANON(YMOUS).

La llamada al sistema munmap borra las ubicaciones para el rango de direcciones especificado, y produce referencias a las direcciones dentro del rango a fin de generar referencias a memoria inválidas.

VALOR DEVUELTO

Si ha funcionado mmap devuelve un puntero al área reservada. En caso de error, es devuelto -1, y errno es modificado apropiadamente. Si ha funcionado munmap devuelve 0, si hay error -1, y errno es fijada (probablemente a EINVAL).

ERRORES

EBADF
fd no es un descriptor de fichero válido (y MAP_ANONYMOUS no ha sido fijado).

EACCES
MAP_PRIVATE fue indicado, pero fd no ha sido abierto para lectura. O MAP_SHARED fue invocado y PROT_WRITE fue fijado, y fd no está abierto para escritura.

EINVAL
No es correcto start o length o offset. (E.g., son demasiado grandes, o no están alineados en los límites de un valor múltiplo de PAGESIZE).

ETXTBUSY
MAP_DENYWRITE fue fijado pero el objeto especificado por fd está abierto para escritura.

EAGAIN
El fichero ha sido bloqueado, o se ha bloqueado una cantidad excesiva de memoria.

ENOMEM
No hay memoria disponible.

Ejemplo[1]

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>

static void
error(char *mesg){
  perror(mesg);
  exit(2);
}

static void
show_usage(const char *prog_name, const char *mesg ){
```

[1] Extraído de: http://www.cs.albany.edu/~maniatty/teaching/networks/examples/csrc/mmap2.c

```
  fprintf(stderr, "Error: %s\n"
    "Usage is:\n%s fname\n\tWhere fname is a file name to use\n",
    mesg, prog_name);
}

int
main(int argc, char *argv[]){

  int fid, child, status;
  pid_t wait_status;
  caddr_t mmap_ptr;
  char buffer[80];

  if (argc != 2){
    show_usage(argv[0], "Wrong number of parameters");
    exit(1);
  }

  /* Create a new file for read/write access with permissions restricted
    to owner rwx access only */

  fid = open(argv[1], O_RDWR | O_CREAT | O_EXCL, (mode_t) 0755 );
  if (fid < 0){
    fprintf(stderr,"Bad Open of file <%s>\n", argv[1]);
    error("Failed to open mmap file, QUIT!");
  }

  status = ftruncate(fid, sizeof(buffer)); /* make the file the buffer size */
  if (status){
      fprintf(stderr,"Could not ftruncate(%d, %d) = %d\n", fid,
        sizeof(buffer), status );
      error("Bad Ftruncate");
  }

  child = fork();
  if (child == -1){
    error("Could not fork, QUIT!");
  } else if (child == 0){ /* This is the child process */

    /* Notice that the child uses the write system call in this version */

    sprintf(buffer, "This is the secret message!");
    status = write(fid, buffer, sizeof(buffer));
    if (status < 0){
      error("Bad Child Write, Quit!");
    }
    status = close(fid); /* close the mmap file */
    if (status == -1){
      error("Bad Child Close, Quit!");
    }
    sleep(2);
```

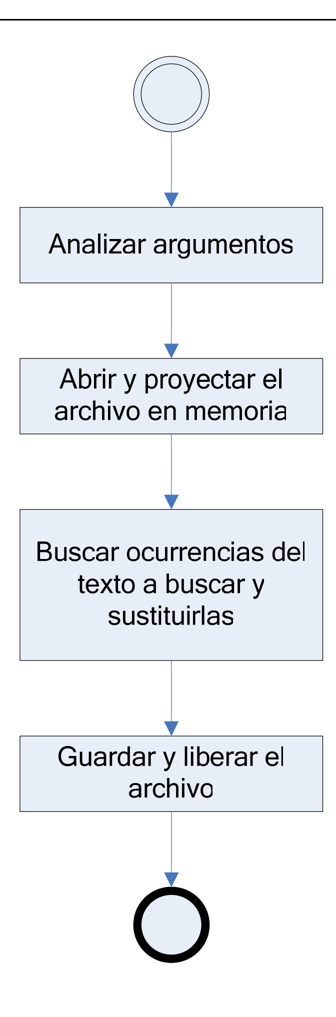
```
  } else { /* This is the parent Process */
   sleep(1);

   /* allocate a shared memory region using mmap which can be inherited
     by child processes */

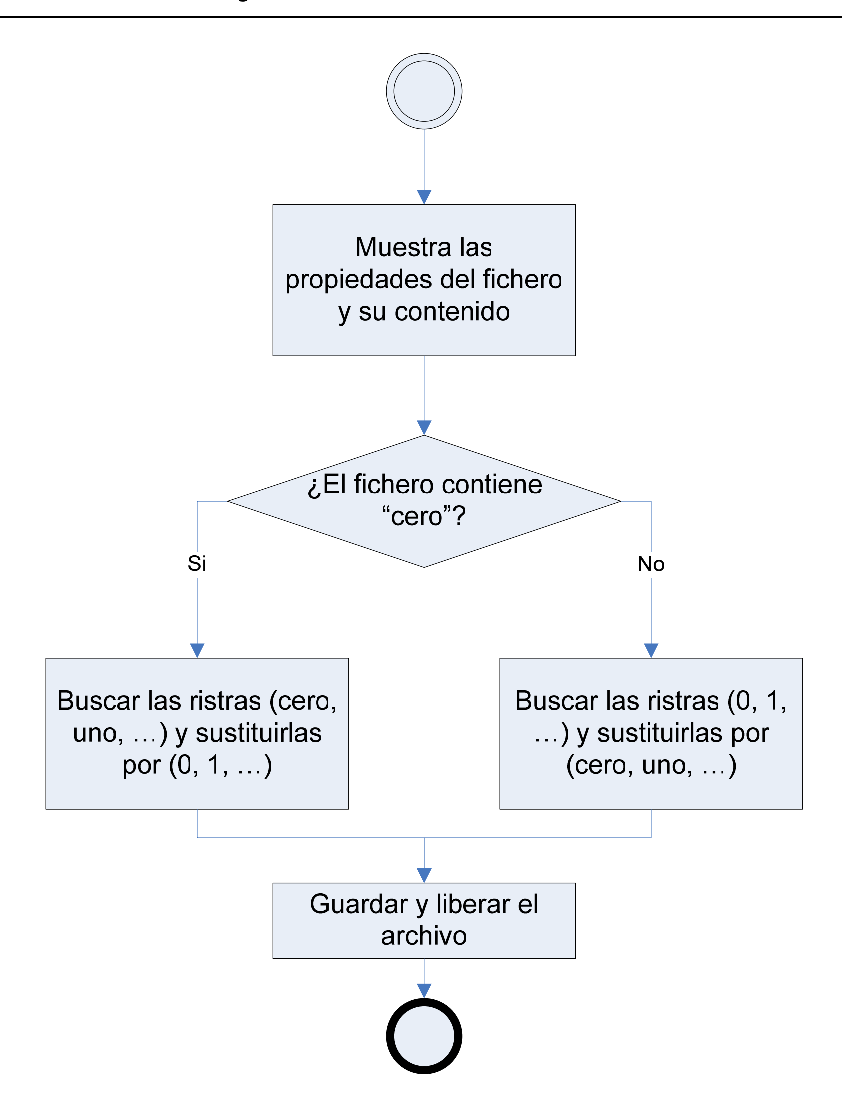
   mmap_ptr = mmap((caddr_t) 0,   /* Memory Location, 0 means O/S chooses */
         sizeof(buffer),/* How many bytes to mmap */
         PROT_READ | PROT_WRITE, /* Read and write permissions */
         MAP_SHARED,    /* Accessible by another process */
         fid,           /* which file is associated with mmap */
         (off_t) 0);    /* Offset in page frame */
   if (mmap_ptr == MAP_FAILED){
     error("Parent Memory Map Failed, QUIT!");
   }
   printf("Parent's mmap_ptr = %lx\n", (unsigned long) mmap_ptr);
   printf("Parent got the message<%s>\n", mmap_ptr);
   status = munmap(mmap_ptr, sizeof(buffer));
   if (status == -1){
     error("Bad munmap, QUIT!");
   }
   status = close(fid); /* close the mmap file */
   if (status == -1){
     error("Bad Parent Close, Quit!");
   }
   status = unlink(argv[1]); /* unlink (i.e. remove) the mmap file */
   if (status == -1){
     error("Bad unlink, QUIT!");
   }
   wait_status = wait( &status );
   if (wait_status == -1){
     error("Bad Wait, QUIT!");
   }
  }
  return 0;
}
```

III. Diagramas de bloques

Programa principal



Guión de Bash adjunto



IV. Descripción de los ficheros y código fuente

map.c

int main (int argn, char ** argv)

Función principal que hace los pasos de tomar argumentos (fichero, ristra a buscar y ristra con que reemplazar), abrir y mapear fichero, aplicar los cambios y salvar el fichero. Todo está hecho llamando funciones en la medida de lo posible.

Parámetros posibles:

- --help → Nos muestra la ayuda del programa
- --f [fichero] → Introduce el fichero a editar y mapear en memoria
- --r [r1] [r2] → Busca las existencias de la ristra “r1” y las cambia por la ristra “r2”.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ristras.h"
#include "mapear.h"

int main(int argn, char **argv){
    char *nombreFichero = NULL; // Nombre del fichero
    char *r1, *r2;              // Ristra a buscar y a reemplazar


    // PASO 1: Parámetros de entrada
   // Ayuda (--help)

   if ((argn == 2) && (strcmp(argv[1],"--help") == 0)){
       printf("Modo de empleo: ./map [OPCIÓN]\n");
        printf("\t-f fichero; Fichero a editar y mapear en memoria\n");
        printf("\t-r r1 r2; Cambia la ristra 'r1', si existe, por la ristra 'r2'\n");
       exit(0);
   }

   // Toma de Parámetros

   int p; for(p=1; p<argn;) {
       if (strcmp(argv[p],"-f") == 0){
```

```
        // Fichero a editar y mapear en memoria

         p++;
         nombreFichero = (char *)malloc(sizeof(char)*(strlen(argv[p])+1));
         strcpy(nombreFichero,argv[p]); p++;
      }else if ((strcmp(argv[p],"-r") == 0) && (argn >= p+2)){

        // Línea a modificar y nueva línea

         p++; r1 = (char *)malloc(sizeof(char)*(strlen(argv[p])+1));
         strcpy(r1,argv[p]); p++;
        r2 = (char *)malloc(sizeof(char)*(strlen(argv[p])+1));
         strcpy(r2,argv[p]); p++;
     }else{

        // Uso Incorrecto

        printf("map: Forma de uso incorrecta\n");
        printf("Pruebe: ./map --help\n");
        exit(0);
     }
  }
   if(!nombreFichero){

      // Uso Incorrecto

     printf("map: Forma de uso incorrecta\n");
     printf("Pruebe: ./map --help\n");
     exit(0);
  }


   // PASO 2: Abrir y Mapear el fichero en memoria

   size_t longitudInicial; int fd;
   char *direccion = mapearFichero(nombreFichero,&longitudInicial,&fd);


   // PASO 3: Modificar fichero

   strcpy(direccion,reemplazar(direccion,r1,r2));


   // PASO 4: Traer (desmapear) fichero de memoria (con los cambios)

   desmapearFichero(fd,longitudInicial,direccion);

   return 0;
}
```

mapear.c / .h

#define DEBUG_FICHERO

En caso de querer depurar el mapeado del fichero activamos esta característica en tiempo de compilación para que, al llamar a la función 'mapearFichero', se imprima también en pantalla.

char *mapearFichero(char *nombreFichero, size_t *longitud, int *fd)

Abre el fichero 'nombreFichero' como escritura/lectura y lo mapea en memoria. Luego devuelve la longitud en 'longitud' y el descriptor del fichero en 'fd' como parámetros pasados por referencia. La función retorna la dirección de memoria base donde se ha mapeado el fichero.

```
char *mapearFichero(char *nombreFichero, size_t *longitud, int *fd){

   // PASO 1: Abrir fichero como escritura/lectura

   if((*fd = open(nombreFichero,O_RDWR | O_CREAT, S_IRUSR | S_IWUSR)) < 0){
      printf("Error al abrir el fichero\n"); exit(0);
   }
   #ifdef DEBUG_FICHERO
      imprimirFichero(*fd);
   #endif


   // PASO 2: Mapear el fichero en memoria

   struct stat estadisticas;   // Obtener estadísticas del fichero
   if(fstat(*fd,&estadisticas) < 0){
      printf("Error al leer estadísticas del fichero\n"); exit(0);
   }

   *longitud = estadisticas.st_size;       // Nº de bytes a leer (todo el fichero)
   int proteccion = PROT_READ|PROT_WRITE;  // Protección con que mapear las
páginas
   int banderas = MAP_SHARED;              // Tipo de ubicación/mapeo de las páginas
   int desplazamiento = 0;                 // Posición del fichero, desde la que tomar los
bytes indicados por longitud (desplazamiento en el marco de página)
   char *memoria;
   if((memoria = mmap(0,*longitud,proteccion,banderas,*fd,desplazamiento)) ==
MAP_FAILED){
      printf("Error al mapear el fichero en memoria\n"); exit(0);
```

```
    }
    return memoria;
}
```

int desmapearFichero(int fd, size_t longitudInicial, char *direccion)

Desmapea el fichero cuyo descriptor es 'fd' de memoria. Antes de ello actualiza el tamaño del fichero y la zona de mapeo en memoria apuntada por 'direccion'. Pasamos de tener una longitud 'longitudInicial' a 'strlen(direccion)'.

```
int desmapearFichero(int fd, size_t longitudInicial, char *direccion){

  // Actualiza el tamaño del fichero

    ftruncate(fd,strlen(direccion));

  // Actualiza el tamaño de la memoria de mapeo

    mremap(direccion,longitudInicial,strlen(direccion),0);

  // Traer/Desmapear el fichero de memoria

    if(munmap(direccion,strlen(direccion)) == -1){
        printf("Error al traer (desmapear) el fichero de memoria\n"); exit(0);
    }
    return 0;
}
```

int imprimirFichero(int f)

Imprime el fichero 'f' por pantalla sin alterar la posición actual del cursor, para ello la guardamos al principio y la restauramos al final.

```
int imprimirFichero(int f){
    off_t p = lseek(f,0,SEEK_CUR);  // Se salva la posición dentro del fichero
    char c;
    int n = read(f,&c,1);
    printf("\e[H\e[2J");    // Borra la pantalla (equivalente al clear)
    while(n){   // Si (n == 0) ==> EOF
        printf("%c",c);
        n = read(f,&c,1);
    }
    lseek(f,p,SEEK_SET);            // Se restaura la posición dentro del fichero
    return 0;
}
```

ristras.c / .h

char *reemplazar(char *ristra, char *buscar, char *reemplazar)

En la ristra 'ristra', reemplaza las ocurrencias que aparecen en la ristra 'buscar' con la ristra 'reemplazar' y devuelve la cadena resultante. Para ello seguimos una estrategia simple, basada en buscar una ocurrencia de 'buscar' en 'ristra', copiar lo anterior a la ocurrencia en el vector de resultado, añadirle 'reemplazar' y volver a repetir los pasos para el resto de la cadena hasta que no hayan más ocurrencias. Cuando no hayan más coincidencias se añade el resto de la ristra si existe.

```
#include <stdlib.h>
#include <string.h>

char *reemplazar(char *ristra, char *buscar, char *reemplazar){

   // ristraAuxiliar --> Almacena el resto de la ristra (que representa el contenido
original del fichero) que falta por analizar
   // coincidencia --> Apunta a la coincidencia. Permite poner '\0' en su primera
posición para tomar lo que hay antes de la coincidencia
   // resultado --> Va acumulando la ristra resultante (que representa el nuevo
contenido del fichero)

   char *ristraAuxiliar = (char *)malloc(sizeof(char)*(strlen(ristra)+1));
strcpy(ristraAuxiliar,ristra);
   char *coincidencia = (char *)malloc(sizeof(char)*(strlen(ristra)+1));
   char *resultado = (char *)malloc(sizeof(char)*(strlen(ristra)+1));

   while(ristraAuxiliar){

     // Buscar coincidencia (la primera por la izquierda)

      coincidencia = strstr(ristraAuxiliar,buscar);
      if(coincidencia){

         // Si hay coincidencia se almacena lo anterior a la coincidencia y la ristra
'reemplazar' y se actualiza la ristraAuxiliar

         coincidencia[0] = '\0'; // Permite copiar lo que hay antes de la coincidencia
         if(!strlen(resultado)) strcpy(resultado,ristraAuxiliar);
         else strcat(resultado,ristraAuxiliar);
         strcat(resultado,reemplazar);
         strcpy(ristraAuxiliar,coincidencia+strlen(buscar)); // Actualizar ristraAuxiliar
      }else{
```

```
        // Si no hay coincidencia, se almacena el resto (ristraAuxiliar actual) y se termina

        if(!strlen(resultado)) strcpy(resultado,ristraAuxiliar);
        else strcat(resultado,ristraAuxiliar);
        break;
      }
    }
    free(ristraAuxiliar); free(coincidencia);
    return resultado;
}
```

Makefile

Hemos almacenado como variables el compilador con sus argumentos, el programa con sus fuentes y un ejemplo de los parámetros de nuestro programa.

Por otro lado, existen 3 reglas definidas:

- ✓ clean → Borra el programa ejecutable.
- ✓ build-map → Borra el programa ejecutable (llamando a clean) y compila nuestro programa.
- ✓ map → Es la regla por defecto. Borra el programa ejecutable (llamando a clean), compila nuestro programa y lo ejecuta con un ejemplo pasado por las variables 'FICHERO', 'RISTRA_BUSCAR' y 'RISTRA_REEMPLAZAR'.

```
#############
# Variables #
#############

COMPILADOR = gcc
OPCIONES = -Wall
PROGRAMA = map
FUENTES = $(PROGRAMA).c ristras.c mapear.c
FICHERO = fichero
RISTRA_BUSCAR = tres
RISTRA_REEMPLAZAR = dos
ARGUMENTOS = -f $(FICHERO) -r $(RISTRA_BUSCAR)
$(RISTRA_REEMPLAZAR)

##########################################################################
###
```

```
# Reglas (usar 'make -s' para que no se muestre el eco de los comandos) #
###############################################################################
###

map: build-map
    echo "Contenido del fichero" $(FICHERO) "original:"; cat $(FICHERO)
    ./$(PROGRAMA) $(ARGUMENTOS)
    echo "Contenido del fichero" $(FICHERO) "modificado:"; cat $(FICHERO)

build-map: clean
    $(COMPILADOR) -o $(PROGRAMA) $(FUENTES) $(OPCIONES)

clean:
    rm -f $(PROGRAMA)
```

V. Ejemplo de uso

Uso general

Para el uso del programa se tienen dos alternativas:

1. Lanzarlo con el Makefile ('make -s' para que no haya eco).

En este caso se debe fijar los valores de la variables FICHERO, RISTRA_BUSCAR y RISTRA_REEMPLAZAR, que por defecto valen:

FICHERO = fichero

RISTRA_BUSCAR = tres

RISTRA_REEMPLAZAR = dos

Al lanzar 'make -s' se compila y lanza el programa './map' con los parÃ¡metros anteriores.

2. Compilar con 'make build-map' y ejecutar './map':

Primero se tiene que compilar, si se han hecho cambios en los fuentes, con la regla 'make build-map' del fichero ‘Makefile’.

Luego se lanza el programa con './map'; los parámetros a lanzar pueden consultarse con './map --help'.

```
make build-map
./map
```

Adicionalmente se dispone de un script llamado 'numeros' que cambia números escritos con ristras a números escritos con cifras, y viceversa. Actúa sobre el fichero 'fichero' creado a tal efecto con números escritos con ristras incialmente.

```
./numeros
```

Guión de ejemplo: numeros

Ejemplo de guión de Bash[2] en el que usamos nuestro programa para modificar las apariciones de los números del 0 al 9 como ristras (cero, uno, dos, …) por su equivalente numérico (0, 1, 2, …) o viceversa.

Su diagrama de bloques está en la sección de "diagramas de bloques" de esta memoria.

La variable 'fichero', declarada al principio inicialmente con el valor 'fichero', representa el fichero a modificar, por lo que deberá ser configurada en caso de querer usar un fichero diferente.

El guión ejecuta los siguientes pasos:

1. Muestra las propiedades y contenido del fichero a modificar.
2. Busca la aparición de la palabra "cero" en el fichero.
3. En caso de encontrarla, sustituye las ristras (cero, uno, …) por sus equivalentes numéricos (0, 1, …) en N pasos, tantos como números sustituyamos (especificados en el vector 'numeros' declarado al principio).
4. En caso de no encontrarla hace la sustitución inversa, es decir, de los números (0, 1, …) a sus ristras (cero, uno, …).
5. Finalmente, muestra de las nuevas propiedades y contenido del fichero ya modificado.

```
#! /bin/bash

fichero=fichero
numeros=(cero uno dos tres cuatro cinco seis siete ocho nueve)

clear
echo "Propiedades del fichero '"$fichero"' original:"
ls -l $fichero
echo "Contenido del fichero '"$fichero"' original:"
cat $fichero
echo "Pulse cualquier tecla para continuar (se modificarÃ¡ el fichero)."
read
```

[2] Bash: Intérprete de comandos muy usado en GNU/Linux. http://www.gnu.org/software/bash/bash.html

```
if grep -q cero $fichero; then
   for ((i=0; i<${#numeros[@]}; i++)) do
      ./map -f $fichero -r ${numeros[$i]} $i
   done
else
   for ((i=0; i<${#numeros[@]}; i++)) do
      ./map -f $fichero -r $i ${numeros[$i]}
   done
fi

echo "Propiedades del fichero '"$fichero"' modificado:"
ls -l $fichero
echo "Contenido del fichero '"$fichero"' modificado:"
cat $fichero
```

Fichero de ejemplo: fichero

Es el fichero de prueba usado por el guión anterior ‘numeros’.

```
Fichero de ejemplo para probar el programa 'map':

Números:
	cero
	uno
	dos
	tres
	cuatro
	cinco
	seis
	siete
	ocho
	nueve

Si se ejecuta el script 'numeros' se cambian los números escritos con ristras a números escritos con cifras, y viceversa.
```